ON GAMBLER'S RUIN PROBLEM IN SOME MODELS

A THESIS

SUBMITTED TO THE DEPARTMENT OF MATHEMATICS

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Çiğdem Sevim

July, 2005

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Azer Kerimov(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Ergün Yalçın

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Farhad Husseinov

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute Engineering and Science

ABSTRACT

ON GAMBLER'S RUIN PROBLEM IN SOME MODELS

Çiğdem Sevim
M.S. in Mathematics
Supervisor: Assoc. Prof. Dr. Azer Kerimov
July, 2005

2-player probability games is an important subject of probability theory. In this thesis, we considered four different two-player probability games. We calculated the expected value of stopping times of the games.

Since the rules of the games are complicated and it is not possible to calculate all possible shuffles, we calculated the results using the Monte Carlo method on a computer.

We showed that the expected value of duration of the games are of the order square of number of cards as we expected.

Keywords: Probability Games, Random Numbers, Expected Duration, Monte Carlo Method.

ÖZET

KUMARBAZIN İFLASI PROBLEMİNİN BAZI MODELLERİ ÜZERİNE

Çiğdem Sevim
Matematik, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Azer Kerimov
Temmuz, 2005

İki kişilik olasılık oyunlarıolasılık teorisinin önemli bir konusudur. Bu tezde, dört farklı iki kişilik olasılık oyununu inceledik. Oyunların beklenen bitme sürelerini hesapladık.

Oyunların kuralları komplike olduğundan ve bütün olası dizilişleri hesaplamak mümkün olmadığından, sonuçları Monte Karlo metodunu kullanarak bilgisayarda hesapladık.

Oyunların beklenen bitme sürelerinin beklenilgiği gibi, kart sayısının karesi derecesinde olduğunu gösterdik.

Anahtar sözcükler: Olasılık Oyunları, Rastgele Sayılar, Beklenen Süre, Monte Karlo Yöntemi.

Acknowledgement

I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Azer Kerimov for his excellent guidance, valuable suggestions, and encouragements.

I am also grateful to Assist. Prof. Dr. Emre Sermutlu who always provided me support and guidance.

My special thanks go to my parents Zerrin and Kemal Sevim, and my sister İrem Sevim for their encouragements and supports.

I would also like to thank my friends for their encouragements and supports. I also thank to Serdar Tasel for approving my ideas and his valuable remarks on coding part of my study.

Contents

1 Introduction **1**

2 The Gambler's Ruin Problem **3**

2.1 Statement of the Problem 3

2.2 Solution of the Problem 5

2.2.1 Uniqueness 5

2.2.2 Solution 6

3 The Monte Carlo Method **10**

3.1 The Monte Carlo Method 11

3.2 Two Distinctive Features of the Monte Carlo Method 11

3.3 Accuracy of the Monte Carlo Method 12

3.4 Generating Random Variables on a Computer 16

3.4.1 Arithmetical Pseudo-Random Generators 16

3.4.2 Physical Random Number Generators 17

3.4.3 Generating Pseudo-Random Numbers Using Visual Basic . 18

4 Games **20**

4.1 Game 1 20

4.2 Game 2 21

4.3 Algorithm of Games 1 and 2 21

4.4 Game 3 22

4.5 Game 4 22

4.6 Algorithm of Games 3 and 4 23

4.7 Results 23

5 Conclusion **28**

A Game 1 **29**

B Game 2 **41**

C Game 3 **53**

D Game 4 **56**

List of Tables

4.1 Results of Game 1 24

4.2 Results of Game 2 25

4.3 Results of Game 3 26

4.4 Results of Game 4 27

Chapter 1

Introduction

The abstract idea of a random walk is that there is some value that randomly goes up, goes down, or stays the same over a sequence of time steps. Many natural phenomena are nicely modelled by random walks.

However, for some reason, random walks are traditionally discussed in the context of some social vice. For example, the value is often regarded as the position of a drunkard who randomly staggers left, staggers right, or just wobbles in place during each time step.

In this thesis, we discuss random walks in the context of gambling. In this case, the value is a gambler's cash-on-hand. This value goes up or down depending on whether he wins or loses each bet.

In second chapter we consider the classical gambler's ruin problem. Consider two players (first and second) who start with respective bankrolls $z\$$ and $(a-z)\$$. Suppose that the second player pays $1\$$ to the first with probability p, and the first player pays to the second with probability q (where $q = 1 - p$). The game stops when the bankroll of the second (first) player reduced to zero. In other words, that player is ruined. We determine the duration of the game until ruin occurs. We examine the relation between the duration of the game, and the initial fortunes. We show that the duration of the game is of order square of the total initial fortunes.

In Chapter 3 we are giving the Monte Carlo method which we use for the calculation of expected duration of the games with complicated rules which are considered in Chapter 4. We examine the same relation for different four models of gambler's ruin problem with complex rules. We are expecting that they have the same relation as the classical gambler's ruin problem.

The rules and the algorithms of the games are given in Chapter 4. We suppose that each player has n different cards $(1, 2, \ldots, n)$. Each player shuffles the cards and starts to compare them. We have two different tables that show which player gets the cards. In the first and second games, the winner put the cards to the bottom of the deck. In the third and fourth games we have the same rules as the first and the second correspondingly. In these games, players shuffle the cards after each move. The games are coded in Visual Basic and the necessary calculations are done using a computer. Also we give the results that we obtain by using the Monte Carlo method in Chapter 4.

Chapter 2

The Gambler's Ruin Problem

2.1 Statement of the Problem

We consider here a typical problem which can be solved by using the total probability formula. We consider a game in which the first player wins 1 unit each time with probability p and loses with probability $1 - p$ independently of the other moves. The game stops when the fortune of the gambler becomes zero (gambler's ruin) or $a > 0$ (gambler's victory) where a is the sum of the fortunes of the gamblers at the beginning of the game. Assume that initial fortune of the gambler is z, $0 < z < a$. The course of the game can be represented conveniently by a graph consisting of straight segments with angels of $\pm 45°$. Each graph starts at the point z, and ends either at $z = a$ (victory) or at $z = 0$ (ruin). It is convenient to assume that after attaining $z = 0$ or a the graph continues as a straight horizontal line. These graphs correspond to the points ω of a space of elementary outcomes Ω. In order to indicate the dependence on the initial point z we write ω_z and Ω_z. For $\omega_z \in \Omega_z$ we set $p(\omega_z) = p^k q^\ell$ where k (or ℓ) is the number of segments which move upwards (or downwards). This definition does not imply that $\sum_{\omega_z \in \Omega_z} p(\omega_z) = 1$. The violation of this last equality can be interpreted as the fact that there is a positive probability for an infinite gain. [2]

Let us denote by W_z the probability of victory and L_z the probability of ruin with initial fortune of z. For $\omega_z \in \Omega_z$ we set

$$P_z = \sum_{\omega_z \in \Omega_z} p(\omega_z), \quad 0 < z < a. \tag{2.1}$$

Also we set

$$\begin{aligned} W_z &= \sum{}' p(\omega_z) \\ & \qquad\qquad 0 < z < a \\ L_z &= \sum{}'' p(\omega_z) \end{aligned} \tag{2.2}$$

where $\sum'$ or $\sum''$ indicates that the summation is carried out for those ω_z that end with a or with 0 correspondingly. Evidently,

$$\begin{aligned} P_0 = 1, &\quad P_a = 1; \\ W_0 = 0, &\quad W_a = 1; \\ L_0 = 1, &\quad L_a = 0. \end{aligned}$$

Lemma 1 *For $0 < z < a$ the following relations hold:*

$$\begin{aligned} P_z &= pP_{z+1} + qP_{z-1}, \quad P_0 = 1, \quad P_a = 1; \\ W_z &= pW_{z+1} + qW_{z-1}, \quad W_0 = 0, \quad W_a = 1; \\ L_z &= pL_{z+1} + qL_{z-1}, \quad L_0 = 1, \quad L_a = 0. \end{aligned} \tag{2.3}$$

Proof Since

$$P_z = \sum_{\omega_z \in \Omega_z} p(\omega_z),$$

we may divide summation into two parts; the sum over those ω_z in which the first game was won, and the sum over those ω_z in which the first game was lost. Therefore

$$P_z = p \sum_{\omega_{z+1} \in \Omega_{z+1}} p(\omega_{z+1}) + q \sum_{\omega_{z-1} \in \Omega_{z-1}} p(\omega_{z-1}).$$

We have

$$P_z = pP_{z+1} + qP_{z-1}.$$

The remaining relations can be showed in the same way. [11], [10]

2.2 Solution of the Problem

We shall find explicit formulas for W_z and L_z. We need the solutions of the homogeneous linear equation

$$R_z = pR_{z+1} + qR_{z-1}, \quad 0 < z < a \tag{2.4}$$

subject to the boundary conditions $R_0 = C_1, R_a = C_2$.

2.2.1 Uniqueness

Lemma 2 *There exists at most one solution to (2.4) which satisfies the given boundary conditions.*

Proof Assume that there are two solutions, $R_z^{(1)}$, $R_z^{(2)}$. Let

$$R_z^{(3)} = R_z^{(1)} - R_z^{(2)}.$$

Then $R_z^{(3)}$ satisfies (2.4) and

$$R_0^{(3)} = R_a^{(3)} = 0.$$

Let us choose z_0 such that $0 < R_{z_0}^{(3)} = \max\{R_z^{(3)}, 0 < z < a\}$. Since

$$R_{z_0}^{(3)} \geq R_{z_0+1}^{(3)},$$

and

$$R_{z_0}^{(3)} \geq R_{z_0-1}^{(3)},$$

(2.4) is possible if and only if

$$R_{z_0}^{(3)} = R_{z_0+1}^{(3)} = R_{z_0-1}^{(3)}.$$

In exactly same way we can go to $z_0 \pm 2$ and so forth. In the end we obtain that $R_z^{(3)}$ =constant for $0 < z < a$. But this constant can only be 0. An analogous

argument can be carried out in the case where $\min R_z^{(3)} < 0$. In the same way we obtain $R_z^{(3)} \equiv 0$ for $0 \leq z \leq a$.

P_z satisfies (2.4) and the boundary conditions $P_0 = P_a = 1$. By lemma the solution $P_z \equiv 1$ is unique. Hence, the probability of an infinite gain is equal to zero.

2.2.2 Solution

We first try particular solutions of the form $R_z = \lambda^z$. The equation (2.4) becomes

$$\lambda = p\lambda^2 + q.$$

So we have solutions

$$\lambda_1 = 1, \lambda_2 = \frac{q}{p}.$$

First case: $p \neq q$
In this case $\lambda_1 \neq \lambda_2$.

$$W_z = d_1{\lambda_1}^z + d_2{\lambda_2}^z = d_1 + d_2(\frac{q}{p})^z,$$

and

$$W_0 = d_1 + d_2 = 0$$

$$W_a = d_1 + d_2(\frac{q}{p})^a = 1$$

Therefore we obtain

$$W_z = \frac{1 - (\frac{q}{p})^z}{1 - (\frac{q}{p})^a}. \tag{2.5}$$

We obtain L_z from

$$L_z = 1 - W_z = \frac{(\frac{p}{q})^{a-z} - 1}{(\frac{p}{q})^a - 1}. \tag{2.6}$$

Second case: $p = q = 1/2$
In this case the equation has the form

$$W_z = \frac{1}{2}(W_{z+1} + W_{z-1}),$$

and also $\lambda_1 = \lambda_2 = 1$. The particular solutions are $W_z = 1$ and $W_z = z$. Therefore the solution is of the form

$$W_z = d_1 + d_2 z.$$

From boundary conditions

$$\begin{aligned} W_0 &= d_1 = 0 \\ W_a &= d_2 a = 1, d_2 = a^{-1}. \end{aligned}$$

Thus

$$W_z = \frac{z}{a} \tag{2.7}$$

and

$$L_z = 1 - W_z = \frac{a - z}{a}. \tag{2.8}$$

Definition 1 *Let $\tau(\omega_z)$, $\omega_z \in \Omega_z$, be an integer-valued random variable which is equal to the duration of the game, i.e. first attaining 0 or a. Then $\tau(\omega_z)$ is called the stopping time.*

Let us consider the expectation of the stopping time:

$$E(\tau(\omega_z)) = \sum_{\omega_z \in \Omega_z} \tau(\omega_z) p(\omega_z)$$

Lemma 3 *Set $E_z = E(\tau(\omega_z))$, then*

$$E_z = pE_{z+1} + qE_{z-1} + 1, \quad 0 < z < a, \quad E_0 = E_a = 0. \tag{2.9}$$

Proof

$$E_z = \sum \tau(\omega_z) p(\omega_z) = \sum_{\omega_z \in B_+} \tau(\omega_z) p(\omega_z) + \sum_{\omega_z \in B_-} \tau(\omega_z) p(\omega_z)$$

where B_+ (or B_-) denotes the event that the first game was won (or lost). Hence

$$\begin{aligned} E_z &= \sum_{\omega_{z+1}\in\Omega_{z+1}}^{(1)} (\tau(\omega_{z+1})+1)pp(\omega_{z+1}) + \sum_{\omega_{z-1}\in\Omega_{z-1}}^{(2)} (\tau(\omega_{z-1})+1)qp(\omega_{z-1}) \\ &= pE_{z+1} + p + qE_{z-1} + q \\ &= pE_{z+1} + qE_{z-1} + 1. \end{aligned}$$

Therefore we have

$$E_z = pE_{z+1} + qE_{z-1} + 1, \quad 0 < z < a, \quad E_0 = E_a = 0. \tag{2.10}$$

First case: $p \neq q$

We note that the function $E_z = \dfrac{z}{q-p}$ satisfies the equation (2.10) but does not satisfy the boundary conditions. We choose $E_z = \dfrac{z}{q-p} + R_z$ since all $R_z = d_1 + d_2(\dfrac{q}{p})^z$ satisfies $R_z = pR_{z+1} + qR_{z-1}$. From the boundary conditions:

$$d_2 = -d_1$$

and

$$\begin{aligned} \frac{a}{q-p} + d_1(1-(\frac{q}{p})^a) &= 0 \\ d_1 &= \frac{a}{(p-q)(1-(\frac{q}{p})^a)}. \end{aligned}$$

Therefore

$$\begin{aligned} E_z &= \frac{z}{q-p} + \frac{a(1-(\frac{q}{p})^z)}{(p-q)(1-(\frac{q}{p})^a)} \\ E_z &= \frac{a-z-(a(\frac{q}{p})^z + z(\frac{q}{p})^a)}{(p-q)(1-(\frac{q}{p})^a)}. \end{aligned} \tag{2.11}$$

Second case: $p = q = 1/2$

In this case we have

$$E_z = \frac{1}{2}(E_{z+1} + E_{z-1}) + 1.$$

$E_z = -z^2$ satisfies this equation. As in the first case we choose $E_z = -z^2 + d_1 + d_2 z$. From the boundary conditions:

$$d_1 = 0$$

$$-a^2 + d_2 a = 0, \quad d_2 = a.$$

Therefore

$$E_z = -z^2 + az = z(a - z). \tag{2.12}$$

E_z attains its maximum value $\frac{a^2}{4}$ for $z = \frac{a}{2}$. This relation shows that the duration of the game for $z \sim \frac{a}{2}$ is of the order $\frac{a^2}{4}$, i.e. of the order of the square of the length of the interval $[0, a]$. [11], [10], [5]

Chapter 3

The Monte Carlo Method

Let us consider 2-player card games which cannot be solved easily as gambler's ruin problem since the rules of the games are more complicated. In this case we decide who wins according to the numbers written on the cards. We have a table which shows the winning cards in each game before the game starts. To play the game, each player writes the numbers 1 to n on separate slips of paper. Then they draw a card and compare the numbers on the cards. The game continues until one of the players has no card. We will consider 4 different models and we are searching for the expected value of the total number of moves that have been done until the game ends.

There are $n!^2$ combinations which we have to calculate to get the expected value. Since it is not an easy job even for a computer for large n's, instead of checking all the possibilities and taking the average of all ending times we use an estimation. One of the most famous and efficient methods is the Monte Carlo method. According to this method we examine the outcome of only a necessary number of shuffles of the cards besides the error of estimation. [9]

3.1 The Monte Carlo Method

The Monte Carlo method consists of solving various problems of computational mathematics by means of the construction of some random process for each such problem. The required quantities are then determined approximately.

The generally accepted birth date of the Monte Carlo method is 1949. The American mathematicians John von Neumann and Stanislaw Ulam are considered its main originators. They proposed that the researchers use computers to try a statistical approach in physics calculations. The "Monte Carlo" designation, is a reference to the famous casino in Monaco. Its use of randomness and the repetitive nature of the process are analogous to the activities conducted at a casino. The name has endured as the term applied to any mathematical procedure or algorithm that employs statistical sampling and random numbers. [8], [4], [6]

3.2 Two Distinctive Features of the Monte Carlo Method

One advantageous feature of the Monte Carlo method is the simple structure of the computation algorithm. As a rule, a program is written to carry out one random trial. This trial is repeated N times, each trial being independent of the rest, and then the results of all trials are averaged. Therefore, the Monte Carlo method is sometimes called the method of statistical trials.

A second feature of the method is that, as a rule, the error of calculations is proportional to $\sqrt{C/N}$, where C is some constant, and N is the number of trials. Hence, it is clear that to obtain another decimal digit in the result, it is necessary to increase N by a factor of 100.

3.3 Accuracy of the Monte Carlo Method

Let the event A which has a probability p of occurring. Let

$$\xi_i = \begin{cases} 1, & \text{if } A \text{ occurs at the } i\text{th test} \\ 0, & \text{otherwise} \end{cases}$$

Hence the total number of tests in which the event A occurs is

$$L = \sum_{i=1}^{N} \xi_i,$$

where N is the total number of tests. The separate tests are supposed to be independent.

The frequency of occurrence of the event A is L/N, which is a random variable having the mathematical expectation

$$E\left(\frac{L}{N}\right) = \frac{1}{N}E(L) = \frac{1}{N}\sum_{i=1}^{N} E(\xi_i) = \frac{Np}{N} = p$$

and the variance

$$V\left(\frac{L}{N}\right) = \frac{1}{N^2}V(L) = \frac{1}{N^2}\sum_{i=1}^{N} V(\xi_i) = \frac{Np(1-p)}{N^2} = \frac{p(1-p)}{N}$$

According to the Law of Large Numbers the frequency of occurrence of the event A is approximately equal to the probability p. In more precise terms, for every $\epsilon > 0$ and for every $\delta > 0$, there exists a number N of tests, such that with probability greater than $1 - \epsilon$ the frequency of occurrence of the event A will differ from the probability p of the occurrence of this event by less than δ:

$$\left|\frac{L}{N} - p\right| < \delta. \tag{3.1}$$

Since p is the required quantity, and L/N is the approximate value obtained for it by Monte Carlo method, then the difference $\left(\frac{L}{N}\right) - p$ is the error of the

Monte Carlo method. The left-hand side of the inequality (3.1) can be estimated by means of the Chebyshev inequality:

$$\left|\frac{L}{N} - p\right| \leq \sqrt{\frac{p(1-p)}{\epsilon N}}, \tag{3.2}$$

where ϵ is the probability of falsity of the inequality. Hence, the error δ of the Monte Carlo method for the computation of the probability of an event A is of the order

$$\delta \sim \frac{1}{\sqrt{N}}. \tag{3.3}$$

Let a process be modelled in which the value of a certain variable ξ is obtained for each of N independent tests. We assume that this variable has a finite mathematical expectation $E(\xi) = m$ and a variance $V(\xi_i) = \sigma^2$. Then its arithmetic mean

$$\overline{\xi} = \frac{L}{N} = \frac{\sum_{i=1}^{N} \xi_i}{N}$$

is approximately the value of the required mathematical expectation m. The quantity $\overline{\xi}$ is the result of the solution the problem by the Monte Carlo method. The following inequality holds:

$$\delta = |\overline{\xi} - m| \leq \sigma\sqrt{\frac{1}{\epsilon N}}, \tag{3.4}$$

which follows from Chebyshev's inequality. The relation (3.3) holds in this case also.

Note that the variable $L = \xi_1 + \xi_2 + \cdots + \xi_N$ is the sum of a large number of independent random variables. But the same may be said concerning the quantity

$$\overline{\xi} = \frac{\xi_1}{N} + \frac{\xi_2}{N} + \cdots + \frac{\xi_N}{N}.$$

Therefore the law of distribution of the variable $\overline{\xi}$ may be found from the Central Limit Theorem of the Theory of Probability. Let r_α be the quantity such that

$$|\overline{\xi} - m| = r_\alpha$$

with probability α.

The most important case is that in which $\overline{\xi}$ follows the normal law of distribution, i.e. Gaussian law. The fact that the variable $\overline{\xi}$ is distributed almost according to the Gaussian law follows from very general properties of the variable ξ_i. Let us consider the fundamental case in which the distribution of $\overline{\xi}$ is approximately Gaussian. Choosing the confidence level for the estimate of the error as $\alpha = 0.997$, we obtain a variable $r_\alpha = 3\sigma_0$, where σ_0 is the root-mean-square deviation of the variable $\overline{\xi}$.

Remark : (The Rule of Three Sigma)

If ξ is normally distributed with $E(\xi) = m$, $V(\xi) = \sigma^2$, then

$$P\{a \leq \xi \leq b\} = \frac{1}{\sqrt{2\pi}\sigma} \int_a^b e^{-(x-m)^2/2\sigma^2} dx$$

Evidently, $t = \dfrac{x-m}{\sigma}$ is normally distributed with $E(t) = 0$ and $V(t) = 1$. Hence,

$$P\{a \leq \xi \leq b\} = \frac{1}{\sqrt{2\pi}} \int_{t_1}^{t_2} e^{-t^2/2} dt$$

where $t_1 = \dfrac{a-m}{\sigma}$ and $t_2 = \dfrac{b-m}{\sigma}$. Therefore

$$P\{a \leq \xi \leq b\} = \Phi(t_2) - \Phi(t_1) \tag{3.5}$$

where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} dt$$

is the normal distribution.
Assume that $|\xi - m| = 3\sigma$. Using the formula (3.5) with parameters $t_1 = -3$, and $t_2 = 3$ we have

$$P\{m - 3\sigma \leq \xi \leq m + 3\sigma\} = \Phi(3) - \Phi(-3) \simeq 0.997 \tag{3.6}$$

Since

$$\sigma_0 = \frac{\sigma}{\sqrt{N}}$$

the error of the Monte Carlo method satisfies the condition

$$\delta = |\overline{\xi} - m| \leq \frac{3\sigma}{\sqrt{N}} \tag{3.7}$$

Frequently, when it is impossible to estimate the variance beforehand, it will be determined according to the statistical estimate of the variance

$$\Delta = \frac{(\xi_1 - \overline{\xi})^2 + (\xi_2 - \overline{\xi})^2 + \cdots + (\xi_N - \overline{\xi})^2}{N - 1}$$

which is obtained during the process of performing the values of the random variable. Therefore (3.6) becomes

$$\delta = |\overline{\xi} - m| \leq 3\sqrt{\frac{\Delta}{N}}$$

where Δ is the estimation of the variance, and N is the number of trials. Evidently, increasing N decreases the error of Monte Carlo method. [12], [1]

3.4 Generating Random Variables on a Computer

In order to solve problems by the Monte Carlo method, it is necessary to have available sources of random numbers. There are various methods for producing random numbers. The successful solution of the problem depends mainly upon an appropriate choice of the method of generation.
[3]

3.4.1 Arithmetical Pseudo-Random Generators

This first technique which is used widely at present is as follows. A random number is found by a computer program, by means of some recurrence relation. This means that each successive number α_{j+1} is formed from the preceding number α_j by applying some algorithm consisting of arithmetic and logical operations. Such a sequence of numbers is nor random, but nevertheless it may satisfy various statistical criteria of randomness. Hence, such numbers are called *pseudo-random*.

The first algorithm for generating pseudo-random numbers, the mid-square method, was proposed by John von Neumann (1947). In this algorithm, the idea is to start with an arbitrary n-digit number, or *seed*. Squaring the number yields a new number with roughly twice as many digits. The middle n digits of the first answer become the first random number. Now we square the first random number and take out the middle n digits, and so on. However, this algorithm tends to produce sequences that eventually get trapped at a single value or in a short repeating cycle.

The most common method of generating pseudo-random sequences on the computer uses a recursive technique called the *linear congruential generator* first suggested by Lehmer (1949). The sequence of pseudo-random numbers (X_n) is

defined on the set of integers by the recursion formula

$$X_{n+1} = (aX_n + c) \mod m \tag{3.8}$$

where X_0 is an initial non-random seed value and m (modulus), a (multiplier) and c (increment) are parameters that can be adjusted for convenience and to ensure the pseudo-random nature of the sequence. For example, m is frequently taken to be the word size on the computer. Lehmer used this relation with $a = 23$, $c = 0$, and $m = 10^8 + 1$ to get a sequence of eight-digit decimal numbers, with period $5,882,352$. .

The principal defect of pseudo-random numbers is the difficulty of estimating theoretically their statistical properties. Moreover, all sequences of pseudo-random numbers which are generated by program are periodic sequences, and hence very long sequences will not appear to be random even from a practical point of view.

George Marsaglia and Arif Zaman (1991) introduced a new class of random-number generators that produce very long sequences of pseudo-random numbers before the numbers start repeating themselves. This technique is based on series of Fibonacci type. The new generators also proved to be efficient and didn't take up inordinately large amounts of a computer's memory. However, high-quality random number generators can still yield incorrect results under certain circumstances. [8]

3.4.2 Physical Random Number Generators

The second technique for generating random numbers consists of using special devices on the computer which transforms the results of some random physical process into a sequence of binary digits within the machine. The register in which the random numbers are generated is usually assigned an address within the general system of addresses in the computer storage. Then a reference to the random number device (RND) reduces to a reading from that store in the machine. The use of the RND increases the speed of a computation, since at every stage of operation of the computer a new random number appears in a fixed standard

cell.

In the early 1950s the Rand Corporation constructed a million-digit table of random numbers using an electrical "roulette wheel". There are two principal techniques for generating random numbers with taking the advantage of Heisenberg uncertainty principle and quantum effects, say by counting decays of radioactive source or by tapping into electrical noise. According to the physical process employed, random number devices are classified as "radio-activity" or "radio-noise".

A defect of such a method is the risk of instability in random number devices. Both of these methods suffer the defects of slowness and ill-defined distributions. Another disadvantage of using a RND is the impossibility of reproducing exactly the results of the computation of a problem. Reproducibility is needed for debugging codes that use the random numbers and for making correlated or anti-correlated computations.

It is usually considered that if the Monte Carlo method is applied systematically on a computer, then it is preferable to have a tested RND rather than to use pseudo-random numbers.

3.4.3 Generating Pseudo-Random Numbers Using Visual Basic

Visual Basic uses the linear congruential method for pseudo-random number generation in the Rnd() function which is a built-in function. The algorithm (3.8) is used with $m = 2^{24}$, $x_0 = 327680$, $a = 1140671485$, and $c = 12820163$. The expression return the floating point number between 0.0 and 1.0 that is returned by the Rnd() function. We use the following formula to create random numbers between [lower bound, upper bound]:

$$RandNum = (Rnd * upperbound) + lowerbound$$

Note that, by default, the Rnd() function will return the same sequence of pseudo-random numbers each time the program is run. For some purposes this may be

appropriate. For other types of applications, such as games, this may not be appropriate. If a different sequence is required, we use the Randomize statement prior to the first call to Rnd(). This will initialize the random number seed by using the system timer.

Chapter 4

Games

Here, we give the rules of the games and construct their algorithms for programming in Visual Basic using the Monte Carlo method. [7]

In the first two games, each player has n cards on their hands:

$\boxed{1}\,\boxed{2}\,\boxed{3}\ldots\boxed{\text{n}}$, $\qquad$ $\boxed{1}\,\boxed{2}\,\boxed{3}\ldots\boxed{\text{n}}$

Let $n > 1$, $1 \leq k \leq n$:

4.1 Game 1

Rules

1. $k < n$ wins each of the numbers: $1, \ldots, k-1$;
2. 1 wins n;
3. After each move the player who wins the cards arranges them in order (better card goes first) and puts them to the bottom of the deck.

4.2 Game 2

Rules

1. $2k$ wins each of the numbers: $2k-1, \ldots, k$; $\forall k = 1, \ldots, n$; if n is even;
2. $2k+1$ wins $2k, \ldots, k$; $\forall k = 1, \ldots, n$; if n is odd;
3. After each move the player who wins the cards arranges them in order (better card goes first) and puts them to the bottom of the deck.

4.3 Algorithm of Games 1 and 2

Step 1 User enters the number of cards n, and the sample size N

Step 2 Declaration of the rules of the game

Step 3 Input: $move = 0$, $sum = 0$, $count = 0$, $Mean = 0$, $Error = 0$

Step 4 Shuffle the decks

Step 5 Each player plays the card on the top of the deck on their hand

Step 6 $move = move + 1$

Step 7 If the cards are not the same, then the one who has the better card according to Step 2 gets the cards on the desk then go to next step. If the cards are same, then go to Step 5

Step 8 Player puts the cards which he/she won to the bottom of the deck in the following order: first the better card, next the others

Step 9 If player1 and player2 still have cards on their hands, then go to Step 5, If not go to next step

Step 10 The player which has the all of the cards wins the game, if all of the cards are on the desk, then we say that the game is tie, $count = count + 1$

Step 11 $Sum = Sum + move$, Save the value of move as $X(i)$ for each $i \leq N$

Step 12 If $count < N$ go to Step 4, If not go to next step

Step 13 $Mean = Sum/N$, For each i $Error = Error + [Mean - X(i)]^2$, $Error = Error/[N(N-1)]$, $Error = 3 * (Error)^{(1/2)}$

Step 14 Output $Mean$, and $Error$

In games 3 and 4, each player has n cards in a box:

4.4 Game 3

Rules

1. $k < n$ wins each of the numbers: $1, \ldots, k-1$;
2. 1 wins n;
3. After each move the players shuffle the cards in the boxes together with the cards that are won.

4.5 Game 4

Rules

1. $2k$ wins each of the numbers: $2k-1, \ldots, k$; $\forall k = 1, \ldots, n$; if n is even;
2. $2k+1$ wins $2k, \ldots, k$; $\forall k = 1, \ldots, n$; if n is odd;
3. After each move the players shuffle the cards in the boxes together with the cards that are won.

4.6 Algorithm of Games 3 and 4

Step 1 User enters the number of cards n, and the sample size N

Step 2 Declaration of the rules of the game

Step 3 Input: $move = 0$, $sum = 0$, $count = 0$, $Mean = 0$, $Error = 0$

Step 4 Shuffle the decks

Step 5 Draw one card from each box

Step 6 $move = move + 1$

Step 7 If the cards are not the same, then the one who has the better card according to Step 2 gets the cards on the desk, then go to next step. If they are same, then go to Step 5

Step 8 If both boxes are not empty, then go to Step 5, If one of the boxes is empty go to next step

Step 9 The player which has the all of the cards wins the game, if all of the cards are on the desk then we say that the game is tie, $count = count + 1$

Step 10 $Sum = Sum + move$, Save the value of move as $X(i)$ for each $i \leq N$

Step 11 If $count < N$ go to Step 4, If not go to next step

Step 12 $Mean = Sum/N$, For each i $Error = Error + [Mean - X(i)]^2$, $Error = Error/[N(N-1)]$, $Error = 3 * (Error)^{(1/2)}$

Step 13 Output $Mean$, and $Error$

4.7 Results

We have calculated the expected value of the duration of the games for a necessary number of shuffles of each games with $2, \ldots, 50$ number of cards. The following tables show the results of each game where mean is the average stopping time of

games calculated using Monte Carlo method besides the error of estimation. If we examine the results we conclude that the durations of the games are of the order n^2 as in the classcal gambler's ruin problem.

Table 4.1: Results of Game 1

# of cards	**Mean**	**Error**	26	451	1.8
2	3	0	27	558	2.7
3	4	0	28	525	3
4	7	0	29	640	3.5
5	15	0	30	604	2.7
6	21	0	31	732	3.6
7	36	0	32	689	3.2
8	41	0.3	33	826	3.1
9	63	0.6	34	777	3.7
10	66	0.5	35	927	3.5
11	95	0.9	36	874	3.6
12	94	0.7	37	1031	3.6
13	133	1.2	38	973	3.6
14	130	1	39	1145	3.7
15	176	1.6	40	1081	3.6
16	169	1.3	41	1262	4.4
17	225	2	42	1195	4
18	215	1.7	43	1391	4.8
19	279	2	44	1316	4.4
20	265	2	45	1512	5.2
21	341	2.2	46	1437	4.8
22	321	1.9	47	1651	5.7
23	407	2.6	48	1565	5.3
24	384	2	49	1790	6.2
25	479	2.3	50	1700	5.8

Table 4.2: Results of Game 2

# of cards	**Mean**	**Error**	26	458	2.3
2	3	0	27	505	2.5
3	4	0	28	527	2.3
4	7	0	29	578	2.6
5	12	0	30	599	2.6
6	19	0	31	652	2.7
7	28	0.1	32	678	2.7
8	38	0.3	33	734	3
9	54	0.5	34	755	3
10	66	0.7	35	815	3.3
11	84	1	36	840	3.4
12	97	1.1	37	904	3.6
13	121	1.4	38	927	3.7
14	135	1.5	39	993	4
15	161	1.8	40	1020	3.9
16	178	2	41	1091	4.7
17	207	2.3	42	1116	4.8
18	225	2	43	1189	5.1
19	257	2.1	44	1218	5.2
20	276	2.3	45	1294	5.5
21	314	2.3	46	1318	5.6
22	332	2.1	47	1403	6
23	373	2.2	48	1426	6.1
24	394	2.2	49	1510	6.5
25	437	2.4	50	1541	6.6

Table 4.3: Results of Game 3

# of cards	**Mean**	**Error**	26	732	3
2	3	0.1	27	791	3.1
3	5	0.1	28	857	3.2
4	11	0.2	29	929	3.4
5	19	0.4	30	996	4
6	31	0.5	31	1064	4
7	44	0.7	32	1134	4.2
8	60	0.9	33	1210	4.2
9	78	0.9	34	1269	4
10	98	1	35	1350	4.4
11	121	1.2	36	1434	4.7
12	145	1.3	37	1557	7
13	173	1.6	38	1628	7.1
14	202	1.6	39	1698	7.3
15	234	1.9	40	1790	7.8
16	270	1.9	41	1875	8.2
17	303	1.9	42	1970	8.6
18	344	2	43	2079	9.2
19	384	1.9	44	2100	9.3
20	432	1.9	45	2268	10.2
21	473	2.1	46	2378	10.5
22	517	2.3	47	2459	10.4
23	568	2.5	48	2597	11.3
24	627	2.7	49	2730	11.7
25	680	2.7	50	2815	11.9

Table 4.4: Results of Game 4

# of cards	**Mean**	**Error**	26	643	2.9
2	3	0.1	27	685	3
3	5	0.1	28	756	3.4
4	11	0.2	29	817	3.7
5	18	0.3	30	875	3.6
6	28	0.5	31	927	3.9
7	40	0.7	32	985	4.1
8	54	0.8	33	1069	4.2
9	70	0.9	34	1115	4.1
10	88	1	35	1175	4.2
11	108	1.2	36	1284	4.7
12	130	1.4	37	1326	6.6
13	154	1.4	38	1425	7.1
14	181	1.6	39	1451	7
15	207	1.8	40	1576	7.7
16	237	2	41	1660	8.2
17	272	2.1	42	1683	8.3
18	301	1.9	43	1803	8.8
19	337	2	44	1868	9
20	381	2.1	45	1984	9.5
21	417	2	46	2039	10
22	459	2.2	47	2131	10.6
23	504	2.4	48	2208	10.8
24	553	2.5	49	2373	11.8
25	595	2.7	50	2451	12.1

Chapter 5

Conclusion

In this work, we considered expected stopping times of four different models of gambler's ruin problem. Because of the complicated rules of the models we used Monte Carlo Method to have approximate solutions. Our aim was to see the relation between the duration of the games and the number of the cards. We started with 2 cards on each player's hand and increased the number of cards one by one up to 50.

Since it was not possible to calculate each $n!^2$ different ways of arranging of n cards, instead of checking all the possibilities we examined the outcome of a necessary number of shuffles which were randomly chosen. The known Monte Carlo method is used to get a random hand and to estimate the results.

We showed that the mean value of the duration of the games are of the order n^2.

Appendix A

Game 1

Here, we are giving the codes of the games written in Visual Basic language. Since Game 3 and 4 are same as Game 1 and 2 correspondingly except the replacement we give the whole codes for Game 1 and 2 but we give only the distinct parts of Game 3 and 4.

```
Private Sub Check1_Click()
If Check1.Value Then
Text4.Enabled = False
Text4.BackColor = &H8000000F
Label20.Caption = "Maxgame="
Else
Text4.Enabled = True
Text4.BackColor = &H80000005
Label20.Caption = "Maxpart="
End If
End Sub
Private Sub Command1_Click()
Dim n As Integer, t As Integer
Dim r As Long, w As Long
Dim dat As String, outdat As String
```

```
n = Val(Text1.Text)
If n < 2 Then
    MsgBox "N must be equal to 2 at least.", vbOKOnly, "Error"
    Exit Sub
End If
Command1.Enabled = False
Text1.Enabled = False
Label3.Caption = "Working"
' Create file 1 for n=1
Open "1.dat" For Binary As 1
outdat = Chr(1)
Put #1, , outdat
Close 1
' Generate combinations up to n
For t = 2 To n
' Generate by using the file t-1
' Create files
Open Mid(Str(t - 1), 2) + ".dat" For Binary As 2
Open Mid(Str(t), 2) + ".dat" For Binary As 1
' Using Induction
For r = 1 To fact(t - 1)
    dat = String(t - 1, " ")
    Get #2, , dat
    For w = 1 To t
        outdat = ""
        If w <> 1 Then outdat = Mid(dat, 1, w - 1)
        outdat = outdat + Chr(t)
        If w <> t Then outdat = outdat + Mid(dat, w)
        Put #1, , outdat
    Next w
' Refresh the progress bar
ProgressBar1.Value = (100 * r) / fact(t - 1)
DoEvents
```

```
Next r
' Close files
Close 1
Close 2
Next t
Label3.Caption = "Ready"
Text1.Enabled = True
Command1.Enabled = True
End Sub
Private Sub Command2_Click()
Dim n As Integer
Dim p1num As Integer, p2num As Integer
Dim bound As Double, p1hand As Long, p2hand As Long
Dim p1start As Long, p2start As Long, p1max As Long, p2max As Long
Dim move As Double, sum As Double, game As Double, infloop As Double
Dim p1wins As Double, p2wins As Double, tie As Double,
Dim part As Double, maxpart As Double
Dim p1 As String, p2 As String, middle As String
Dim p1dup As String, p2dup As String
Dim mean As Double
Dim rndmode As Boolean
Dim mainloop As Long, ml As Long
Dim pval As Integer
Dim error As Double
Dim d As Long
n = Val(Text2.Text)
bound = Val(Text3.Text)
part = Val(Text4.Text)
maxpart = Val(Text5.Text)
If n < 2 Then
    MsgBox "N must be equal to 2 at least.", vbOKOnly, "Error"
    Exit Sub
End If
```

```
If bound < 1 Then
    MsgBox "Bound must be a positive integer.", vbOKOnly, "Error"
    Exit Sub
End If
Text2.Enabled = False
Text3.Enabled = False
Command2.Enabled = False
Label9.Caption = "Working"
Label7.Caption = "-"
Label12.Caption = "-"
Label16.Caption = "-"
Label17.Caption = "-"
Label18.Caption = "-"
Label22.Caption = "-"
logfn = Replace(Date$ + "_" + Time$ + ".txt", ":", ".", , ,
vbBinaryCompare)
Open logfn For Output As 3
Print #3, "Started at " + Time$ + " on " + Date$
Print #3, "--------------------------------"
Print #3, ""
' Start Game
sum = 0
mean = 0
game = 0
infloop = 0
p1wins = 0
p2wins = 0
tie = 0
rndmode = Check1.Value
mainloop = 1
If rndmode Then
mainloop = maxpart
ReDim xgame(maxpart) As Long
```

```
End If
'--------------------------------
If Not rndmode Then
' Open combination file n for Player 1
Open Mid(Str(n), 2) + ".dat" For Binary As 1
' Open combination file n for Player 2
Open Mid(Str(n), 2) + ".dat" For Binary As 2
End If
Print #3, "N =" + Str(n)
If rndmode Then
Print #3, "Random Mode = Yes"
Print #3, "Maxgame =" + Str(mainloop)
Else
Print #3, "Random Mode = No"
Print #3, "Part =" + Str(part) + " /" + Str(maxpart)
End If
Print #3, "Bound =" + Str(bound)
Print #3, ""
For ml = 1 To mainloop
If rndmode Then
    p1start = 1
    p2start = 1
    p1max = 1
    p2max = 1
    Else
    p1start = 1 + (((part - 1) * fact(n)) / maxpart)
    p2start = 1
    p1max = p1start + (fact(n) / maxpart) - 1
    p2max = p2start + fact(n) - 1
End If
'--------------------------------
For p1hand = p1start To p1max
If rndmode Then
```

```
p1 = GetRandomHand(n)
Else
    p1 = String(n, " ")
    Get #1, (p1hand - 1) * n + 1, p1
End If
p1dup = p1
For p2hand = p2start To p2max
If rndmode Then
p2 = GetRandomHand(n)
Else
    p2 = String(n, " ")
    Get #2, (p2hand - 1) * n + 1, p2
End If
p2dup = p2

' Start Playing
' Middle is empty
middle = ""
move = 0
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
 + BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
Do
' Player1 and then Player2 play
middle = Left(p1, 1) + Left(p2, 1) + middle

If Len(p1) = 1 Then p1 = "" Else p1 = Mid(p1, 2)
If Len(p2) = 1 Then p2 = "" Else p2 = Mid(p2, 2)
move = move + 1
```

```
p1num = Asc(Mid(middle, 1, 1))
p2num = Asc(Mid(middle, 2, 1))
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
'Rules
If Not (p1num = p2num) Then
    If (((p1num = n) And (p2num = 1)) Or ((p1num = 1) And (p2num = n))) Then
    If ((p1num = 1) And (p2num = n)) Then
    ' P1 wins
    p1 = p1 + middle
    middle = ""
    End If
    If ((p1num = n) And (p2num = 1)) Then
    ' P2 wins
    temp = Mid(middle, 1, 1)
    Mid(middle, 1, 1) = Mid(middle, 2, 1)
    Mid(middle, 2, 1) = temp
    p2 = p2 + middle
    middle = ""
    End If
    Else
    If p2num < p1num Then
    ' P2<P1 => P1 wins
    p1 = p1 + middle
    middle = ""
    Else
    ' P2>P1 => P2 wins
    temp = Mid(middle, 1, 1)
```

```
    Mid(middle, 1, 1) = Mid(middle, 2, 1)
    Mid(middle, 2, 1) = temp
    p2 = p2 + middle
    middle = ""
    End If
    End If
End If
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :"
+ BinToStr(middle), vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
DoEvents
Loop While ((Not (p1 = "" Or p2 = "")) And (move <= bound))
' Results
' Count if bounded
If (move <= bound) Then
    If (p1 = "" And p2 <> "") Or (p1 <> "" And p2 = "") Then
    sum = sum + move
    game = game + 1
    If p1 = "" Then
        p2wins = p2wins + 1
        Else
        p1wins = p1wins + 1
    End If
    Else
        ' Count tie case?
            sum = sum + move
            game = game + 1
        tie = tie + 1
    End If
```

```
    Else
    infloop = infloop + 1
    ' Log Infinity log case
    Print #3, "NonEnding Game :"
    Print #3, "Player 1 :"
    Print #3, BinToStr(p1dup)
    Print #3, "Player 2 :"
    Print #3, BinToStr(p2dup)
    Print #3, ""
End If
' Set player1's hand as beginning
p1 = p1dup
' Refresh the progress bar
If rndmode Then
pval = (100 * ml) / mainloop
Else
pval = ((((fact(n) * (p1hand - p1start)) + p2hand) * 100)
/ (fact(n) ^ 2)) * maxpart
End If
ProgressBar2.Value = pval
Next p2hand
Next p1hand
If rndmode Then
    If move > bound Then
    xgame(ml) = -1
    Else
    xgame(ml) = move
    End If
End If
Next ml
If Not rndmode Then
' Close the file for player 2
Close 2
```

```
' Close the file for player 1
Close 1
End If
' End Game
If rndmode Then
        If Not game = 0 Then mean = sum / mainloop
        ' Calculating Error
        error = 0
        For d = 1 To mainloop
        If Not xgame(d) = -1 Then error = error + ((mean - xgame(d)) ^ 2)
        Next d
        If game > 1 Then
        error = error / game
        error = error / (game - 1)
        End If
        error = 3 * (error ^ (1 / 2))
        If game > 1 Then
        Label22.Caption = "+- " + Str(error)
        Else
        Label22.Caption = "Infinite"
        End If
Else
    If game <> 0 Then
        mean = sum / (fact(n)) ^ 2
    End If
End If
Label7.Caption = "Average move is" + Str(mean)
Label12.Caption = Str(infloop)
Label16.Caption = Str(p1wins)
Label17.Caption = Str(p2wins)
Label18.Caption = Str(tie)
Print #3, ""
Print #3, "Result :"
```

```
Print #3, "--------"
Print #3, Label7.Caption
Print #3, "Nonending games = " + Label12.Caption
Print #3, "Player1 wins : " + Label16.Caption
Print #3, "Player2 wins : " + Label17.Caption
Print #3, "Tie : " + Label18.Caption
Print #3, "Error = " + Label22.Caption
Print #3, ""
Print #3, "-----------------------------------"
Print #3, "Completed at " + Time$ + " on " + Date$
Close 3
Text2.Enabled = True
Text3.Enabled = True
Command2.Enabled = True
Label9.Caption = "Ready"
End Sub
Private Sub Command3_Click()
Close
End
End Sub
Private Sub Form_Load()
Text1.Text = "6"
ProgressBar1.Value = 0
Label3.Caption = "Ready"
Text2.Text = "2"
Text3.Text = "24"
Label9.Caption = "Ready"
Randomize Timer
End Sub
Private Function fact(n As Integer) As Long
If n < 2 Then
    fact = 1
    Exit Function
```

```
End If
fact = n * fact(n - 1)
End Function
'Define a fnc to get a random hand
Private Function GetRandomHand(n As Integer) As String
Dim x As Integer, r As Integer
Dim temp As String
GetRandomHand = ""
For x = 1 To n
GetRandomHand = GetRandomHand + Chr(x)
Next x
For x = 1 To n
r = Round(Rnd * (n - 1)) + 1
temp = Mid(GetRandomHand, r, 1)
Mid(GetRandomHand, r, 1) = Mid(GetRandomHand, x, 1)
Mid(GetRandomHand, x, 1) = temp
Next x
End Function
Private Function BinToStr(Bin As String) As String
Dim i As Integer
BinToStr = ""
For i = 1 To Len(Bin)
BinToStr = BinToStr + Str(Asc(Mid(Bin, i, 1)))
Next i
End Function
```

Appendix B

Game 2

```
Dim Table(255, 255) As Byte
Private Sub Check1_Click()
If Check1.Value Then
Text4.Enabled = False
Text4.BackColor = &H8000000F
Label20.Caption = "Maxgame="
Else
Text4.Enabled = True
Text4.BackColor = &H80000005
Label20.Caption = "Maxpart="
End If
End Sub
Private Sub Command1_Click()
Dim n As Integer, t As Integer
Dim r As Long, w As Long
Dim dat As String, outdat As String
n = Val(Text1.Text)
If n < 2 Then
    MsgBox "N must be equal to 2 at least.", vbOKOnly, "Error"
    Exit Sub
```

```
End If
Command1.Enabled = False
Text1.Enabled = False
Label3.Caption = "Working"
' Create file 1 for n=1
Open "1.dat" For Binary As 1
outdat = Chr(1)
Put #1, , outdat
Close 1
' Generate combinations up to n
For t = 2 To n
' Generate by using the file t-1
' Create files
Open Mid(Str(t - 1), 2) + ".dat" For Binary As 2
Open Mid(Str(t), 2) + ".dat" For Binary As 1
' Using Induction
For r = 1 To fact(t - 1)
    dat = String(t - 1, " ")
    Get #2, , dat
    For w = 1 To t
        outdat = ""
        If w <> 1 Then outdat = Mid(dat, 1, w - 1)
        outdat = outdat + Chr(t)
        If w <> t Then outdat = outdat + Mid(dat, w)
        Put #1, , outdat
    Next w
' Refresh the progress bar
ProgressBar1.Value = (100 * r) / fact(t - 1)
DoEvents
Next r
' Close files
Close 1
Close 2
```

```
Next t
Label3.Caption = "Ready"
Text1.Enabled = True
Command1.Enabled = True
End Sub
Private Sub Command2_Click()
Dim n As Integer
Dim p1num As Integer, p2num As Integer
Dim bound As Double, p1hand As Long, p2hand As Long
Dim p1start As Long, p2start As Long, p1max As Long, p2max As Long
Dim move As Double, sum As Double, game As Double, infloop As
Double
Dim p1wins As Double, p2wins As Double, tie As Double, part As
Double, maxpart As Double
Dim p1 As String, p2 As String, middle As String
Dim p1dup As String, p2dup As String
Dim mean As Double
Dim rndmode As Boolean
Dim mainloop As Long, ml As Long
Dim pval As Integer
Dim error As Double
Dim d As Long
n = Val(Text2.Text)
bound = Val(Text3.Text)
part = Val(Text4.Text)
maxpart = Val(Text5.Text)
If n < 2 Then
    MsgBox "N must be equal to 2 at least.", vbOKOnly, "Error"
    Exit Sub
End If
If bound < 1 Then
    MsgBox "Bound must be a positive integer.", vbOKOnly, "Error"
    Exit Sub
```

```
End If
Text2.Enabled = False
Text3.Enabled = False
Command2.Enabled = False
Label9.Caption = "Working"
Label7.Caption = "-"
Label12.Caption = "-"
Label16.Caption = "-"
Label17.Caption = "-"
Label18.Caption = "-"
Label22.Caption = "-"
logfn = Replace(Date$ + "_" + Time$ + ".txt", ":", ".", , ,
vbBinaryCompare)
Open logfn For Output As 3
Print #3, "Started at " + Time$ + " on " + Date$
Print #3, "--------------------------------"
Print #3, ""
' Start Game
sum = 0
mean = 0
game = 0
infloop = 0
p1wins = 0
p2wins = 0
tie = 0
CalcTable (n)
rndmode = Check1.Value
mainloop = 1
If rndmode Then
mainloop = maxpart
ReDim xgame(maxpart) As Long
End If
'--------------------------------
```

```
If Not rndmode Then
' Open combination file n for Player 1
Open Mid(Str(n), 2) + ".dat" For Binary As 1
' Open combination file n for Player 2
Open Mid(Str(n), 2) + ".dat" For Binary As 2
End If
Print #3, "N =" + Str(n)
If rndmode Then
Print #3, "Random Mode = Yes"
Print #3, "Maxgame =" + Str(mainloop)
Else
Print #3, "Random Mode = No"
Print #3, "Part =" + Str(part) + " /" + Str(maxpart)
End If
Print #3, "Bound =" + Str(bound)
Print #3, ""
For ml = 1 To mainloop
If rndmode Then
    p1start = 1
    p2start = 1
    p1max = 1
    p2max = 1
    Else
    p1start = 1 + (((part - 1) * fact(n)) / maxpart)
    p2start = 1
    p1max = p1start + (fact(n) / maxpart) - 1
    p2max = p2start + fact(n) - 1
End If
'-------------------------------
For p1hand = p1start To p1max
If rndmode Then
p1 = GetRandomHand(n)
Else
```

```
    p1 = String(n, " ")
    Get #1, (p1hand - 1) * n + 1, p1
End If
p1dup = p1
For p2hand = p2start To p2max
If rndmode Then
p2 = GetRandomHand(n)
Else
    p2 = String(n, " ")
    Get #2, (p2hand - 1) * n + 1, p2
End If
p2dup = p2
' Start Playing
' Middle is empty
middle = ""
move = 0
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
Do
' Player1 and then Player2 play
middle = Left(p1, 1) + Left(p2, 1) + middle
If Len(p1) = 1 Then p1 = "" Else p1 = Mid(p1, 2)
If Len(p2) = 1 Then p2 = "" Else p2 = Mid(p2, 2)
move = move + 1
p1num = Asc(Mid(middle, 1, 1))
p2num = Asc(Mid(middle, 2, 1))
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
```

```
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
'Rules
If Table(p2num, p1num) = 1 Then
    ' P2<P1 => P1 wins
    p1 = p1 + middle
    middle = ""
End If
If Table(p2num, p1num) = 0 Then
    ' P2>P1 => P2 wins
    temp = Mid(middle, 1, 1)
    Mid(middle, 1, 1) = Mid(middle, 2, 1)
    Mid(middle, 2, 1) = temp
    p2 = p2 + middle
    middle = ""
End If
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
DoEvents
Loop While ((Not (p1 = "" Or p2 = "")) And (move <= bound))
' Results
' Count if bounded
If (move <= bound) Then
    If (p1 = "" And p2 <> "") Or (p1 <> "" And p2 = "") Then
    sum = sum + move
```

```
    game = game + 1
    If p1 = "" Then
        p2wins = p2wins + 1
        Else
        p1wins = p1wins + 1
    End If
    Else
        ' Count tie case?
            sum = sum + move
            game = game + 1
            tie = tie + 1
    End If
    Else
    infloop = infloop + 1
    ' Log NonEnding Game
    Print #3, "NonEnding Game :"
    Print #3, "Player 1 :"
    Print #3, BinToStr(p1dup)
    Print #3, "Player 2 :"
    Print #3, BinToStr(p2dup)
    Print #3, ""
End If
' Set player1's hand as beginning
p1 = p1dup
' Refresh the progress bar
If rndmode Then
pval = (100 * ml) / mainloop
Else
pval = ((((fact(n) * (p1hand - p1start)) + p2hand) * 100) /
(fact(n) ^ 2)) * maxpart
End If
ProgressBar2.Value = pval
Next p2hand
```

```
Next p1hand
If rndmode Then
    If move > bound Then
    xgame(ml) = -1
    Else
    xgame(ml) = move
    End If
End If
Next ml
If Not rndmode Then
' Close the file for player 2
Close 2
' Close the file for player 1
Close 1
End If
' End Game
If rndmode Then
        If Not game = 0 Then mean = sum / mainloop
        ' Calculating Error
        error = 0
        For d = 1 To mainloop
        If Not xgame(d) = -1 Then error = error + ((mean -
xgame(d)) ^ 2)
        Next d
        If game > 1 Then
        error = error / game
        error = error / (game - 1)
        End If
        error = 3 * (error ^ (1 / 2))
        If game > 1 Then
        Label22.Caption = "+- " + Str(error)
        Else
        Label22.Caption = "Infinite"
```

```
        End If
Else
    If game <> 0 Then
        mean = sum / (fact(n)) ^ 2
    End If
End If
Label7.Caption = "Average move is" + Str(mean)
Label12.Caption = Str(infloop)
Label16.Caption = Str(p1wins)
Label17.Caption = Str(p2wins)
Label18.Caption = Str(tie)
Print #3, ""
Print #3, "Result :"
Print #3, "--------"
Print #3, Label7.Caption
Print #3, "Nonending games = " + Label12.Caption
Print #3, "Player1 wins : " + Label16.Caption
Print #3, "Player2 wins : " + Label17.Caption
Print #3, "Tie : " + Label18.Caption
Print #3, "Error = " + Label22.Caption
Print #3, ""
Print #3, "-----------------------------------"
Print #3, "Completed at " + Time$ + " on " + Date$
Close 3
Text2.Enabled = True
Text3.Enabled = True
Command2.Enabled = True
Label9.Caption = "Ready"
End Sub
Private Sub Command3_Click()
Close
End
End Sub
```

```
Private Sub Form_Load()
Text1.Text = "6"
ProgressBar1.Value = 0
Label3.Caption = "Ready"
Text2.Text = "2"
Text3.Text = "24"
Label9.Caption = "Ready"
Randomize Timer
End Sub
Private Function fact(n As Integer) As Long
If n < 2 Then
    fact = 1
    Exit Function
End If
fact = n * fact(n - 1)
End Function
'Define a fnc to get a random hand
Private Function GetRandomHand(n As Integer) As String
Dim x As Integer, r As Integer
Dim temp As String
GetRandomHand = ""
For x = 1 To n
GetRandomHand = GetRandomHand + Chr(x)
Next x
For x = 1 To n
r = Round(Rnd * (n - 1)) + 1
temp = Mid(GetRandomHand, r, 1)
Mid(GetRandomHand, r, 1) = Mid(GetRandomHand, x, 1)
Mid(GetRandomHand, x, 1) = temp
Next x
End Function
Private Function BinToStr(Bin As String) As String
Dim i As Integer
```

```
BinToStr = ""
For i = 1 To Len(Bin)
BinToStr = BinToStr + Str(Asc(Mid(Bin, i, 1)))
Next i
End Function
Private Sub CalcTable(n As Integer)
Dim row(255) As Byte
Dim x As Integer, k As Integer
Dim y As Integer
n = n + ((n + 1) Mod 2)
k = (n + 1) / 2
' Setting rows
For x = 1 To n
    If x = 1 Then row(x) = 2
    If x > 1 And x <= k Then row(x) = 1
    If x > k Then row(x) = 0
Next x
' Setting the Table
For y = 1 To n
    For x = 1 To n
    Table(y, ((x + y - 2) Mod n) + 1) = row(x)
    Next x
Next y
End Sub
```

Appendix C

Game 3

This one is slightly different from Game 1. We will change only the playing the game part in Game 1 as follows:

```
For ml = 1 To mainloop '
    p1start = 1 p2start = 1 p1max = 1 p2max = 1
'------------------------------- p1 = GetAllHand(n)
p1dup = p1
p2 = GetAllHand(n)
p2dup = p2
' Start Playing
' Middle is empty
middle = ""
move = 0
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
```

```
Do
' Player1 and then Player2 play
' Choose random
r1 = Int(Rnd * Len(p1)) + 1
r2 = Int(Rnd * Len(p2)) + 1
middle = Mid(p1, r1, 1) + Mid(p2, r2, 1) + middle
p1 = Mid(p1, 1, r1 - 1) + Mid(p1, r1 + 1, Len(p1) - r1)
p2 = Mid(p2, 1, r2 - 1) + Mid(p2, r2 + 1, Len(p2) - r2)
move = move + 1
p1num = Asc(Mid(middle, 1, 1))
p2num = Asc(Mid(middle, 2, 1))
If Check2.Value Then
If Not MsgBox("Player 1 :" + BinToStr(p1) + vbCrLf + "Player 2 :"
+ BinToStr(p2) + vbCrLf + "Middle (p1,p2) :" + BinToStr(middle),
vbOKCancel, "Status") = vbOK Then
Check2.Value = False
End If
End If
If Not (p1num = p2num) Then
    If (((p1num = n) And (p2num = 1)) Or ((p1num = 1) And (p2num =
n))) Then
    If ((p1num = 1) And (p2num = n)) Then
    ' P1 wins
    p1 = p1 + middle
    middle = ""
    End If
    If ((p1num = n) And (p2num = 1)) Then
    ' P2 wins
    p2 = p2 + middle
    middle = ""
    End If
    Else
    If p2num < p1num Then
```

```
    ' P2<P1 => P1 wins
    p1 = p1 + middle
    middle = ""
    Else
    ' P2>P1 => P2 wins
    p2 = p2 + middle
    middle = ""
    End If
    End If
End If
```

In this case we use the following function to get a random hand :

```
Private Function GetAllHand(n As Integer) As String
Dim x As Integer
GetAllHand = ""
For x = 1 To n
GetAllHand = GetAllHand + Chr(x)
Next x
End Function
```

Appendix D

Game 4

This one is slightly different from Game 2. We change only the playing the game part in Game 2 as follows:

```
Do
' Player1 and then Player2 play
' Choose random
r1 = Int(Rnd * Len(p1)) + 1
r2 = Int(Rnd * Len(p2)) + 1
middle = Mid(p1, r1, 1) + Mid(p2, r2, 1) + middle
p1 = Mid(p1, 1, r1 - 1) + Mid(p1, r1 + 1, Len(p1) - r1)
p2 = Mid(p2, 1, r2 - 1) + Mid(p2, r2 + 1, Len(p2) - r2)
move = move + 1
p1num = Asc(Mid(middle, 1, 1))
p2num = Asc(Mid(middle, 2, 1))
```

The getrandomhand function in Game 2 is modified as follows:

```
Private Function GetAllHand(n As Integer) As String
```

```
Dim x As Integer
GetAllHand = ""
For x = 1 To n
GetAllHand = GetAllHand + Chr(x)
Next x
End Function
```

Bibliography

[1] N. P. Buslenko, D. I. Golenko, Yu. A. Shreider, I. M. Sobol, and V. G. Sragovich, *The Monte Carlo method. The method of statistical trials*, Pergamon Press, Oxford, 1966.

[2] Morris H. DeGroot, *Probability and statistics*, Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont., 1975.

[3] Roger Eckhardt, *Stan Ulam, John von Neumann, and the Monte Carlo method*, Los Alamos Sci. (1987), no. 15, Special Issue, With contributions by Tony Warnock, Gary D. Doolen and John Hendricks, Stanislaw Ulam 1909–1984.

[4] Ivar Ekeland, *The broken dice, and other mathematical tales of chance*, University of Chicago Press, Chicago, IL, 1993.

[5] Malcom Goldman, *Introduction to probability and statistics*, Harcourt, Brace & World, Inc., New York, 1970.

[6] Richard Isaac, *The pleasures of probability*, Springer-Verlag, New York, 1995.

[7] Groh Michael Norton, Peter, *Peter norton's guide to visual basic 6*, Sams, Indianapolis, Ind., 1998.

[8] Ivars Peterson, *The jungles of randomness*, John Wiley & Sons Inc., New York, 1998.

[9] A. N. Shiryayev, *Optimal stopping rules*, Springer-Verlag, New York, 1978.

[10] ———, *Probability*, Graduate Texts in Mathematics, vol. 95, Springer-Verlag, New York, 1984.

[11] Yakov G. Sinai, *Probability theory*, Springer-Verlag, Berlin, 1992.

[12] Ilya M. Sobol, *A primer for the Monte Carlo method*, CRC Press, Boca Raton, FL, 1994.